

1  
2  
3  
4

# FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS

5  
6

## FIPA SL Content Language Specification

7

<b>Document title</b>	FIPA SL Content Language Specification		
<b>Document number</b>	XC00008H	<b>Document source</b>	FIPA TC Communication
<b>Document status</b>	Experimental	<b>Date of this status</b>	2002/11/01
<b>Supersedes</b>	FIPA00003		
<b>Contact</b>	fab@fipa.org		
<b>Change history</b>	See <i>Informative Annex B — ChangeLog</i>		

8  
9  
10  
11  
12  
13  
14  
15  
16  
17

18 © 1996-2002 Foundation for Intelligent Physical Agents  
19 <http://www.fipa.org/>  
20 *Geneva, Switzerland*

### Notice

Use of the technologies described in this specification may infringe patents, copyrights or other intellectual property rights of FIPA Members and non-members. Nothing in this specification should be construed as granting permission to use any of the technologies described. Anyone planning to make use of technology covered by the intellectual property rights of others should first obtain permission from the holder(s) of the rights. FIPA strongly encourages anyone implementing any part of this specification to determine first whether part(s) sought to be implemented are covered by the intellectual property of others, and, if so, to obtain appropriate licenses or other permission from the holder(s) of such intellectual property prior to implementation. This specification is subject to change without notice. Neither FIPA nor any of its Members accept any responsibility whatsoever for damages or liability, direct or consequential, which may result from the use of this specification.

## 21 **Foreword**

22 The Foundation for Intelligent Physical Agents (FIPA) is an international organization that is dedicated to promoting the  
23 industry of intelligent agents by openly developing specifications supporting interoperability among agents and agent-  
24 based applications. This occurs through open collaboration among its member organizations, which are companies and  
25 universities that are active in the field of agents. FIPA makes the results of its activities available to all interested parties  
26 and intends to contribute its results to the appropriate formal standards bodies where appropriate.

27 The members of FIPA are individually and collectively committed to open competition in the development of agent-  
28 based applications, services and equipment. Membership in FIPA is open to any corporation and individual firm,  
29 partnership, governmental body or international organization without restriction. In particular, members are not bound to  
30 implement or use specific agent-based standards, recommendations and FIPA specifications by virtue of their  
31 participation in FIPA.

32 The FIPA specifications are developed through direct involvement of the FIPA membership. The status of a  
33 specification can be either Preliminary, Experimental, Standard, Deprecated or Obsolete. More detail about the process  
34 of specification may be found in the FIPA Document Policy [f-out-00000] and the FIPA Specifications Policy [f-out-  
35 00003]. A complete overview of the FIPA specifications and their current status may be found on the FIPA Web site.

36 FIPA is a non-profit association registered in Geneva, Switzerland. As of June 2002, the 56 members of FIPA  
37 represented many countries worldwide. Further information about FIPA as an organization, membership information,  
38 FIPA specifications and upcoming meetings may be found on the FIPA Web site at <http://www.fipa.org/>.

39 **Contents**

40	1	Scope.....	1
41	2	Grammar FIPA SL Concrete Syntax .....	2
42	2.1	Lexical Definitions .....	3
43	3	Notes on FIPA SL Semantics .....	5
44	3.1	Grammar Entry Point: FIPA SL Content Expression.....	5
45	3.2	Well-Formed Formulas.....	5
46	3.3	Atomic Formula .....	6
47	3.4	Terms .....	7
48	3.5	Referential Operators .....	7
49	3.5.1	iota .....	7
50	3.5.2	Any .....	9
51	3.5.3	All .....	10
52	3.6	Functional Terms .....	11
53	3.7	Result Predicate .....	12
54	3.8	Actions and Action Expressions.....	12
55	3.9	Notes on the Grammar Rules .....	12
56	3.10	Representation of Time .....	13
57	4	Reduced Expressivity Subsets of FIPA SL.....	14
58	4.1	FIPA SL0: Minimal Subset .....	14
59	4.2	FIPA SL1: Propositional Form.....	15
60	4.3	FIPA SL2: Decidability Restrictions.....	16
61	5	References .....	19
62	6	Informative Annex A — Syntax and Lexical Notation .....	20
63	7	Informative Annex B — ChangeLog .....	21
64	7.1	2002/11/01 - version H by TC X2S .....	21

65 **1 Scope**

66 This specification defines a concrete syntax for the FIPA Semantic Language (SL) content language. This syntax and  
67 its associated semantics are suggested as a candidate content language for use in conjunction with the FIPA Agent  
68 Communication Language (see [FIPA00037]). In particular, the syntax is defined to be a sub-grammar of the very  
69 general s-expression syntax.  
70

## 71 2 Grammar FIPA SL Concrete Syntax

72 This content language is denoted by the normative constant `fipa-sl` in the `:language` parameter of an ACL  
73 message. See Section 6 for an explanation of the used syntactic notation.

74		
75	Content	= "(" ContentExpression+ ")".
76		
77	ContentExpression	= IdentifyingExpression
78		ActionExpression
79		Proposition.
80		
81	Proposition	= Wff.
82		
83	Wff	= AtomicFormula
84		"(" UnaryLogicalOp Wff ")"
85		"(" BinaryLogicalOp Wff Wff ")"
86		"(" Quantifier Variable Wff ")"
87		"(" ModalOp Agent Wff ")"
88		"(" ActionOp ActionExpression ")"
89		"(" ActionOp ActionExpression Wff ")".
90		
91	UnaryLogicalOp	= "not".
92		
93	BinaryLogicalOp	= "and"
94		"or"
95		"implies"
96		"equiv".
97		
98	AtomicFormula	= PropositionSymbol
99		"(" BinaryTermOp TermOrIE TermOrIE ")"
100		"(" PredicateSymbol TermOrIE+ ")"
101		"true"
102		"false".
103		
104	BinaryTermOp	= "="
105		"result".
106		
107	Quantifier	= "forall"
108		"exists".
109		
110	ModalOp	= "B"
111		"U"
112		"PG"
113		"I".
114		
115	ActionOp	= "feasible"
116		"done".
117		
118	TermOrIE <sup>1</sup>	= Term
119		IdentifyingExpression.
120		
121	Term	= Variable
122		FunctionalTerm
123		ActionExpression
124		Constant
125		Sequence
126		Set.
127		
128	IdentifyingExpression	= "(" ReferentialOperator TermOrIE Wff ")".
129		

<sup>1</sup> Note that this grammar rule is used to group and represent both Terms and Identifying Expressions.

```

130 ReferentialOperator = "iota"
131                       | "any"
132                       | "all".
133
134 FunctionalTerm       = "(" FunctionSymbol TermOrIE* ")"
135                       | "(" FunctionSymbol Parameter* ")" .
136
137 Constant             = NumericalConstant
138                       | String
139                       | DateTime.
140
141 NumericalConstant    = Integer
142                       | Float.
143
144 Variable             = VariableIdentifier.
145
146 ActionExpression     = "(" "action" Agent TermOrIE ")"
147                       | "(" "|" ActionExpression ActionExpression ")"
148                       | "(" ";" ActionExpression ActionExpression ")" .
149
150 PropositionSymbol    = String.
151
152 PredicateSymbol      = String.
153
154 FunctionSymbol       = String.
155
156 Agent                = TermOrIE.
157
158 Sequence             = "(" "sequence" TermOrIE* ")" .
159
160 Set                  = "(" "set" TermOrIE* ")" .
161
162 Parameter            = ParameterName ParameterValue.
163
164 ParameterValue      = TermOrIE.
165
166

```

## 167 2.1 Lexical Definitions

168 All white space, tabs, carriage returns and line feeds between tokens should be skipped by the lexical analyser. See  
 169 Section 6 for an explanation of the used notation.

```

170
171 String                = Word
172                       | ByteLengthEncodedString
173                       | StringLiteral.
174
175 ByteLengthEncodedString = "#" DecimalLiteral+ "\" <byte sequence>.
176
177 Word                  = [~ "\0x00" - "\0x20", "(", ")", "#", "0" - "9", ":", "-", "?"]
178                       [~ "\0x00" - "\0x20", "(", ")]"*.
179
180 ParameterName        = ":" String.
181
182 VariableIdentifier    = "?" String.
183
184 Sign                  = [ "+", "-" ].
185
186 Integer               = Sign? DecimalLiteral+
187                       | Sign? "0" ["x", "X"] HexLiteral+.
188
189 Dot                   = "."
190
191 Float                 = Sign? FloatMantissa FloatExponent?

```

```

192          | Sign? DecimalLiteral+ FloatExponent.
193
194 FloatMantissa = DecimalLiteral+ Dot DecimalLiteral*
195               | DecimalLiteral* Dot DecimalLiteral+.
196
197 FloatExponent = Exponent Sign? DecimalLiteral+.
198
199 Exponent      = ["e", "E"].
200
201 DecimalLiteral = ["0" - "9"].
202
203 HexLiteral     = ["0" - "9", "A" - "F", "a" - "f"].
204
205 StringLiteral  = "\""( [~ "\""]
206                 | "\\\"")*\"".
207
208 DateTime      = Sign? Year Month Day "T" Hour Minute
209               Second MilliSecond TypeDesignator?.
210
211 Year          = DecimalLiteral DecimalLiteral DecimalLiteral DecimalLiteral.
212
213 Month         = DecimalLiteral DecimalLiteral.
214
215 Day           = DecimalLiteral DecimalLiteral.
216
217 Hour          = DecimalLiteral DecimalLiteral.
218
219 Minute        = DecimalLiteral DecimalLiteral.
220
221 Second        = DecimalLiteral DecimalLiteral.
222
223 MilliSecond   = DecimalLiteral DecimalLiteral DecimalLiteral.
224
225 TypeDesignator = ["a" - "z" , "A" - "Z"].
226

```

## 227 3 Notes on FIPA SL Semantics

228 This section contains explanatory notes on the intended semantics of the constructs introduced in above.  
229

### 230 3.1 Grammar Entry Point: FIPA SL Content Expression

231 An FIPA SL content expression may be used as the content of an ACL message. There are three cases:  
232

- 233 • A proposition, which may be assigned a truth value in a given context. Precisely, it is a well-formed formula (Wff) using the rules described in the `wff` production. A proposition is used in the `inform` communicative act (CA) and other CAs derived from it.  
234  
235
- 236 • An action, which can be performed. An action may be a single action or a composite action built using the sequencing and alternative operators. An action is used as a content expression when the act is `request` and other CAs derived from it.  
237  
238
- 239 • An identifying reference expression (IRE), which identifies an object in the domain. This is the Referential operator and is used in the `inform-ref` macro act and other CAs derived from it.  
240  
241  
242  
243

244 Other valid content expressions may result from the composition of the above basic cases. For instance, an action-condition pair (represented by an `ActionExpression` followed by a `wff`) is used in the `propose` act; an action-condition-reason triplet (represented by an `ActionExpression` followed by two `wffs`) is used in the `reject-proposal` act. These are used as arguments to some ACL CAs in [FIPA00037].  
245  
246  
247  
248

### 249 3.2 Well-Formed Formulas

250 A well-formed formula is constructed from an atomic formula, whose meaning will be determined by the semantics of the underlying domain representation or recursively by applying one of the construction operators or logical connectives described in the `wff` grammar rule. These are:  
251  
252

- 253 • `(not <wff>)`  
254 Negation. The truth value of this expression is false if `wff` is true. Otherwise it is true.  
255  
256
- 257 • `(and <wff0> <wff1>)`  
258 Conjunction. This expression is true iff<sup>2</sup> well-formed formulae `wff0` and `wff1` are both true, otherwise it is false.  
259
- 260 • `(or <wff0> <wff1>)`  
261 Disjunction. This expression is false iff well-formed formulae `wff0` and `wff1` are both false, otherwise it is true.  
262
- 263 • `(implies <wff0> <wff1>)`  
264 Implication. This expression is true if either `wff0` is false or alternatively if `wff0` is true and `wff1` is true. Otherwise it is false. The expression corresponds to the standard material implication connective `wff0 implies wff1`.  
265  
266
- 267 • `(equiv <wff0> <wff1>)`  
268 Equivalence. This expression is true if either `wff0` is true and `wff1` is true, or alternatively if `wff0` is false and `wff1` is false. Otherwise it is false.  
269  
270
- 271 • `(forall <variable> <wff>)`  
272 Universal quantification. The quantified expression is true if `wff` is true for every value of value of the quantified variable.  
273  
274
- 275 • `(exists <variable> <wff>)`

---

<sup>2</sup> If and only if.



276 Existential quantification. The quantified expression is true if there is at least one value for the variable for which  
 277 `Wff` is true.

278

279 • `(B <agent> <expression>)`

280 Belief. It is true that `agent` believes that `expression` is true.

281

282 • `(U <agent> <expression>)`

283 Uncertainty. It is true that `agent` is uncertain of the truth of `expression`. `Agent` neither believes `expression`  
 284 nor its negation, but believes that `expression` is more likely to be true than its negation.

285

286 • `(I <agent> <expression>)`

287 Intention. It is true that `agent` intends that `expression` becomes true and will plan to bring it about.

288

289 • `(PG <agent> <expression>)`

290 Persistent goal. It is true that `agent` holds a persistent goal that `expression` becomes true, but will not  
 291 necessarily plan to bring it about.

292

293 • `(feasible <ActionExpression> <Wff>)`

294 It is true that `ActionExpression` (or, equivalently, some event) can take place and just afterwards `Wff` will be  
 295 true.

296

297 • `(feasible <ActionExpression>)`

298 Same as `(feasible <ActionExpression> true)`.

299

300 • `(done <ActionExpression> <Wff>)`

301 It is true that `ActionExpression` (or, equivalently, some event) has just taken place and just before that `Wff` was  
 302 true.

303

304 • `(done <ActionExpression>)`

305 Same as `(done <ActionExpression> true)`.

306

### 307 3.3 Atomic Formula

308 The atomic formula represents an expression which has a truth value in the language of the domain of discourse. Three  
 309 forms are defined:

310

311 • A given propositional symbol may be defined in the domain language, which is either true or false,

312

313 • Two terms may or may not be equal under the semantics of the domain language, or,

314

315 • Some predicate is defined over a set of zero or more arguments, each of which is a term.

316

317 The FIPA SL representation does not define a meaning for the symbols in atomic formulae: this is the responsibility of  
 318 the domain language representation and ontology. Several forms are defined:

319

320 • `true false`

321 These symbols represent the true proposition and the false proposition.

322

323 • `(= Term1 Term2)`

324 `Term1` and `Term2` denote the same object under the semantics of the domain.

325

326 Other predicates may be defined over a set of arguments, each of which is a term, by using the `(PredicateSymbol`  
 327 `Term+)` production.

328

329 The FIPA SL representation does not define a meaning for other symbols in atomic formulae: this is the responsibility of  
 330 the domain language representation and the relative ontology.  
 331

### 332 3.4 Terms

333 Terms are either themselves atomic (constants and variables) or recursively constructed as a functional term in which a  
 334 functor is applied to zero or more arguments. Again, FIPA SL only mandates a syntactic form for these terms. With  
 335 small number of exceptions (see below), the meanings of the symbols used to define the terms are determined by the  
 336 underlying domain representation.  
 337

338 Note that, as mentioned above, no legal well-formed expression contains a free variable, that is, a variable not declared  
 339 in any scope within the expression. Scope introducing formulae are the quantifiers (`forall`, `exists`) and the  
 340 reference operators `iota`, `any` and `all`. Variables may only denote terms, not well-formed formulae.  
 341

### 342 3.5 Referential Operators

#### 343 3.5.1 `iota`

- 344 • (`iota <term> <formula>`)

345 The `iota` operator introduces a scope for the given expression (which denotes a term), in which the given  
 346 identifier, which would otherwise be free, is defined. An expression containing a free variable is not a well-formed  
 347 FIPA SL expression. The expression (`iota x (P x)`) may be read as “the `x` such that `P` [is true] of `x`”. The `iota`  
 348 operator is a constructor for terms which denote objects in the domain of discourse.  
 349

350 Notice that, unlike a term, an identifying expression can have different interpretations by different agents because  
 351 its formal definition depends on the KB.  
 352

- 353 • **Formal Definition**

354 A `iota` expression can only be evaluated with respect to a given theory. Suppose KB is a knowledge base such  
 355 that  $T(KB)$  is the theory generated from KB by a given reasoning mechanism. Formally,  $\iota(\tau, \phi) = \theta\tau$  iff  $\theta\tau$  is a term  
 356 that belongs to the set  $\Sigma = \{\theta\tau : \theta\phi \in T(KB)\}$  and  $\Sigma$  is a singleton; or  $\iota(\tau, \phi)$  is undefined if  $\Sigma$  is not a singleton. In this  
 357 definition  $\theta$  is a most general variable substitution,  $\theta\tau$  is the result of applying  $\theta$  to  $\tau$ , and  $\theta\phi$  is the result of applying  
 358  $\theta$  to  $\phi$ . This implies that a failure occurs if no object or more than one object satisfies the condition specified in the  
 359 `iota` operator.  
 360

361 If  $\iota(\tau, \phi)$  is undefined then any term, identifying expression or well-formed formula containing  $\iota(\tau, \phi)$  is also  
 362 undefined.  
 363

- 364 • **Example 1**

365 This example depicts an interaction between agent A and B that makes use of the `iota` operator, where agent A is  
 366 supposed to have the following knowledge base  $KB = \{P(A), Q(1, A), Q(1, B)\}$ .  
 367

```

368 (query-ref
369   :sender (agent-identifier :name B)
370   :receiver (set (agent-identifier :name A))
371   :content
372     "((iota ?x (p ?x)))"
373   :language fipa-sl
374   :reply-with query1)
375
376 (inform
377   :sender (agent-identifier :name A)
378   :receiver (set (agent-identifier :name B))
379   :content
380     "((= (iota ?x (p ?x)) a)) "
381   :language fipa-sl
```

```
382 :in-reply-to query1)
```

383  
384 The only object that satisfies proposition  $P(x)$  is  $a$ , therefore, the `query-ref` message is replied by the `inform`  
385 message as shown.

386  
387 • **Example 2**

388 This example shows another successful interaction but more complex than the previous one.

```
389 (query-ref
390 :sender (agent-identifier :name B)
391 :receiver (set (agent-identifier :name A))
392 :content
393   "((iota ?x (q ?x ?y)))"
394 :language fipa-sl
395 :reply-with query2)
396
397 (inform
398 :sender (agent-identifier :name A)
399 :receiver (set (agent-identifier :name B))
400 :content
401   "((= (iota ?x (q ?x ?y)) 1))"
402 :language fipa-sl
403 :in-reply-to query2)
404
```

405  
406 The most general substitutions  $\theta$  such that  $\theta Q(x, y)$  can be derived from KB are  $\theta_1=\{x/1, y/A\}$  and  $\theta_2=\{x/1, y/B\}$ .  
407 Therefore, the set  $\Sigma=\{\theta\tau: \theta\phi\in T(KB)\}=\{\{x/1, y/A\}x, \{x/1, y/B\}x\}=\{1\}$  is a singleton and hence `(iota ?x (q ?x ?y))`  
408 represents the object 1.

409  
410 • **Example 3**

411 Finally, this example shows an unsuccessful interaction using the `iota` operator. In this case, agent A cannot  
412 evaluate the `iota` expression and therefore a failure message is returned to agent B

```
413 (query-ref
414 :sender (agent-identifier :name B)
415 :receiver (set (agent-identifier :name A))
416 :content
417   "((iota ?y (q ?x ?y)))"
418 :language fipa-sl
419 :reply-with query3)
420
421 (failure
422 :sender (agent-identifier :name A)
423 :receiver (set (agent-identifier :name B))
424 :content
425   "((action (agent-identifier :name A)
426     (inform-ref
427       :sender (agent-identifier :name A)
428       :receiver (set (agent-identifier :name B))
429       :content
430         \"((iota ?y (q ?x ?y)))\"
431       :language fipa-sl
432       :in-reply-to query3)))"
433   more-than-one-answer)
434 :language fipa-sl
435 :in-reply-to query3)
436
```

437  
438 The most general substitutions that satisfy  $Q(x, y)$  are  $\theta_1=\{x/1, y/a\}$  and  $\theta_2=\{x/1, y/b\}$ , therefore, the set  $\Sigma=\{\theta\tau:$   
439  $\theta\phi\in T(KB)\}=\{\{x/1, y/A\}y, \{x/1, y/B\}y\}=\{A, B\}$ , which is not a singleton. This means that the `iota` expression used in  
440 this interaction is not defined.

441

442 **3.5.2 Any**

- 443 • (any <term> <formula>)

444 The any operator is used to denote any object that satisfies the proposition represented by formula.

445

446 Notice that, unlike a term, an identifying expression can have different interpretations by different agents because  
447 its formal definition depends on the KB.

448

449 • **Formal Definition**450 An any expression can only be evaluated with respect to a given theory. Suppose KB is a knowledge base such  
451 that T(KB) is the theory generated from KB by a given reasoning mechanism. Formally, any( $\tau$ ,  $\phi$ )= $\theta\tau$  iff  $\theta\tau$  is a term  
452 that belongs to the set  $\Sigma=\{\theta\tau: \theta\phi\in T(KB)\}$ ; or any( $\tau$ ,  $\phi$ ) is undefined if  $\Sigma$  is the empty set. In this definition  $\theta$  is a most  
453 general variable substitution,  $\theta\tau$  is the result of applying  $\theta$  to  $\tau$ , and  $\theta\phi$  is the result of applying  $\theta$  to  $\phi$ .

454

455 If the set  $\Sigma$  is empty then any term, identifying expression or well-formed formula containing any( $\tau$ ,  $\phi$ ) is undefined.

456

457 If the set  $\Sigma$  is not empty, then for any formula  $\psi$  containing any( $\tau$ ,  $\phi$ ) let  $\psi'$  be the formula obtained from  $\psi$  by  
458 replacing any( $\tau$ ,  $\phi$ ) with a variable  $x$  (not occurring in  $\psi$ ) and let  $s\_k$  be a new Skolem constant. Then  $\psi$  is true when  
459  $\{x/s\_k\}\psi'$  element\_of T(KB union  $\{\tau/s\_k\}\phi$ ),  $\psi$  is false when  $\{x/s\_k\}\text{not}(\psi')$  element\_of T(KB union  $\{\tau/s\_k\}\phi$ ), and  
460 otherwise  $\psi$  is undefined.

461

462 In other words if  $\psi$  contains any( $\tau$ ,  $\phi$ ),  $\psi$  is true if a modified form of  $\psi$  obtained by replacing the any expression in it  
463 with a new constant  $s\_k$  can be inferred based on the assumption that  $\phi$  holds of  $s\_k$ .  $\psi$  is false if  $\text{not}(\psi)$  inferred  
464 in a similar way. This definition is needed to avoid the following contradiction:

465

466 (implies  
467 (and (= Stephen (any ?x (fipa-member ?x)))  
468 (= Farooq (any ?x (fipa-member ?x))))  
469 (= Stephen Farooq))

470

471 This definition implies that failures only occur if there are no objects satisfying the condition specified as the second  
472 argument of the any operator.

473

474 If any( $\tau$ ,  $\phi$ ) is undefined then any term, identifying expression or well-formed formula containing any( $\tau$ ,  $\phi$ ) is also  
475 undefined.

476

477 • **Example 4**478 Assuming that agent A has the following knowledge base KB={P(A), Q(1, A), Q(1, B)}, this example shows a  
479 successful interaction with agent A using the any operator.

480

481 (query-ref  
482 :sender (agent-identifier :name B)  
483 :receiver (set (agent-identifier :name A))  
484 :content  
485 "(any (sequence ?x ?y) (q ?x ?y))"  
486 :language fipa-sl  
487 :reply-with query1)  
488  
489 (inform  
490 :sender (agent-identifier :name A)  
491 :receiver (set (agent-identifier :name B))  
492 :content  
493 "(= (any (sequence ?x ?y) (q ?x ?y)) (sequence 1 a))"  
494 :language fipa-sl  
495 :in-reply-to query1)

496

The most general substitutions  $\theta$  such that  $\theta Q(x, y)$  can be derived from KB are  $\{x/1, y/A\}$  and  $\{x/1, y/B\}$ , therefore  $\Sigma = \{\theta \text{Sequence}(x, y) : \theta Q(x, y) \in T(\text{KB})\} = \{\text{Sequence}(1, A), \text{Sequence}(1, B)\}$ . Using this set, agent A chooses the first element of  $\Sigma$  as the appropriate answer to agent B.

- **Example 5**

This example shows an unsuccessful interaction with agent A, using the `any` operator.

```
(query-ref
  :sender (agent-identifier :name B)
  :receiver (set (agent-identifier :name A))
  :content
    "((any ?x (r ?x)))"
  :language fipa-sl
  :reply-with query2)

(failure
  :sender (agent-identifier :name A)
  :receiver (set (agent-identifier :name B))
  :content
    "((action (agent-identifier :name A)
      (inform-ref
        :sender (agent-identifier :name A)
        :receiver (set (agent-identifier :name B))
        :content
          \"((any ?x (r ?x)))\"
        :language fipa-sl
        :in-reply-to query2))
      (unknown-predicate r)))"
  :language fipa-sl
  :in-reply-to query2)
```

Since agent A does not know the `r` predicate, the answer to the query that had been sent by agent B cannot be determined, therefore a failure message is sent to agent B from agent A. The failure message specifies the failure's reason (that is, `unknown-predicate r`)

### 3.5.3 All

- `(all <term> <formula>)`

The `all` operator is used to denote the set of all objects that satisfy the proposition represented by `formula`.

Notice that, unlike a term, an identifying expression can have different interpretations by different agents because its formal definition depends on the KB.

- **Formal Definition**

An `all` expression can only be evaluated with respect to a given theory. Suppose KB is a knowledge base such that  $T(\text{KB})$  is the theory generated from KB by a given reasoning mechanism. Formally,  $\text{all}(\tau, \phi) = \{\theta\tau : \theta\phi \in T(\text{KB})\}$ . Notice that  $\text{all}(\tau, \phi)$  may be a singleton or even an empty set. In this definition  $\theta$  is a most general variable substitution,  $\theta\tau$  is the result of applying  $\theta$  to  $\tau$ , and  $\theta\phi$  is the result of applying  $\theta$  to  $\phi$ .

If no objects satisfy the condition specified as the second argument of the `all` operator, then the identifying expression denotes an empty set.

- **Example 6**

Suppose agent A has the following knowledge base  $\text{KB} = \{P(A), Q(1, A), Q(1, B)\}$ . This example shows a successful interaction between agent A and B that make use of the `all` operator.

```
(query-ref
  :sender (agent-identifier :name B)
```

```

554     :receiver (set (agent-identifier :name A))
555     :content
556     "((all (sequence ?x ?y) (q ?x ?y)))"
557     :language fipa-sl
558     :reply-with query1)
559
560 (inform
561   :sender (agent-identifier :name A)
562   :receiver (set (agent-identifier :name B))
563   :content
564   "((= (all (sequence ?x ?y) (q ?x ?y)) (set(sequence 1 a)(sequence 1 b))))"
565   :language fipa-sl
566   :in-reply-to query1)
567

```

568 The set of the most general substitutions  $\theta$  such that  $\theta Q(x, y)$  can be derived from KB is  $\{\{x/1, y/A\}, \{x/1, y/B\}\}$ ,  
 569 therefore  $\text{all}(\text{Sequence}(x, y), Q(x, y)) = \{\text{Sequence}(1, A), \text{Sequence}(1, B)\}$ .

570

- 571 **Example 7**

572 Following Example 6, if there is no possible answer to a query making use of the `all` operator, then the agent  
 573 should return the empty set.

```

574
575 (query-ref
576   :sender (agent-identifier :name B)
577   :receiver (set (agent-identifier :name A))
578   :content
579   "((all ?x (q ?x c)))"
580   :language fipa-sl
581   :reply-with query2)
582
583 (inform
584   :sender (agent-identifier :name A)
585   :receiver (set (agent-identifier :name B))
586   :content
587   "((= (all ?x (q ?x c))(set)))"
588   :language fipa-sl
589   :in-reply-to query2)
590

```

591 Since there is no possible substitution for  $x$  such that  $Q(x, C)$  can be derived from KB, then  $\text{all}(x, Q(x, c)) = \{\}$ . In this  
 592 interaction the term `(set)` represents the empty set.

593

### 594 3.6 Functional Terms

595 A functional term refers to an object via a functional relation (referred by the `FunctionSymbol`) with other objects (that  
 596 is, the terms or parameters), rather than using the direct name of that object, for example, `(fatherOf Jesus)` rather  
 597 than `God`.

598

599 Two syntactical forms can be used to express a functional term. In the first form the functional symbol is followed by a  
 600 list of terms that are the arguments of the function symbol. The semantics of the arguments is position-dependent, for  
 601 example, `(divide 10 2)` where 10 is the dividend and 2 is the divisor. In the second form each argument is preceded  
 602 by its name, for example, `(divide :dividend 10 :divisor 2)`. The encoder is required to adopt the following  
 603 criteria to select which form to use in order to represent a functional term. The first form, that is, the position-dependent  
 604 form, should be used to encode all those functional terms for which the ontology does not specify the names of the  
 605 parameters (for example, all the functions of the `fipa-agent-management` ontology). The second form, that is, the  
 606 parameter-name dependent form, must be used to encode all those functional terms for which the ontology does  
 607 specify the names of the parameters but not their position (for example, all the object descriptions of the `fipa-agent-`  
 608 `management` ontology). This second form is particularly appropriate to represent descriptions where the function  
 609 symbol should be interpreted as the constructor of an object, while the parameters represent the attributes of the object.

610

611 The following is an example of an object, instance of a vehicle class:

```

612
613 (vehicle
614   :colour red
615   :max-speed 100
616   :owner (Person
617     :name Luis
618     :nationality Portuguese))
619

```

620 Some ontologies may decide to give a description of some concepts only in one or both of these two forms, that is by  
 621 specifying, or not, a default order to the arguments of each function in the domain of discourse. How this order is  
 622 specified is outside the scope of this specification.

623  
 624 Functional terms can be constructed by a domain functor applied to zero or more terms.  
 626

### 627 3.7 Result Predicate

628 A common need is to determine the result of performing an action or evaluating a term. To facilitate this operation, a  
 629 standard predicate `result`, of arity two, is introduced to the language. `result/2` has the declarative meaning that the  
 630 result of evaluating a term, or equivalently of performing an action, encoded by the first argument term, is the second  
 631 argument term. However, it is expected that this declarative semantics will be implemented in a more efficient,  
 632 operational way in any given FIPA SL interpreter.

633  
 634 A typical use of the `result` predicate is with a variable scoped by `iota`, giving an expression whose meaning is, for  
 635 example, "the `x` which is the result of agent `i` performing `act`":

```

636
637 (iota x (result (action i act) x)))
638

```

### 639 3.8 Actions and Action Expressions

640 Action expressions are a special subset of terms. An action itself is introduced by the keyword `action` and comprises  
 641 the agent of the action (that is, an identifier representing the agent performing the action) and a term denoting the action  
 642 which is [to be] performed.

643  
 644 Notice that a specific type of action is an ACL communicative act (CA). When expressed in FIPA SL, syntactically an  
 645 ACL communicative act is an action where the agent of the action is the `sender` of the CA, and the term denotes the  
 646 CA including all its parameters where the performative should be used as a function symbol, as referred by the used  
 647 ontology. Example 5 includes an example of an ACL CA, encoded as a `String`, whose content embeds another CA.

648  
 649 Two operators are used to build terms denoting composite CAs:  
 650

- 651 • The sequencing operator (`;`) denotes a composite act in which the first action (represented by the first operand) is  
 652 followed by the second action, and,
- 653  
 654 • The alternative operator (`|`) denotes a composite act in which either the first action occurs, or the second, but not  
 655 both.  
 656

### 657 3.9 Notes on the Grammar Rules

- 658 1. The standard definitions for integers and floating point are assumed. However, due to the necessarily unpredictable  
 659 nature of cross-platform dependencies, agents should not make strong assumptions about the precision with which  
 660 another agent is able to represent a given numerical value. FIPA SL assumes only 32-bit representations of both  
 661 integers and floating point numbers. Agents should not exchange message contents containing numerical values  
 662 requiring more than 32 bits to encode precisely, unless some prior arrangement is made to ensure that this is valid.  
 663
- 664 2. All keywords are case-insensitive.

665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686

3. A length encoded string is a context sensitive lexical token. Its meaning is as follows: the message envelope of the token is everything from the leading # to the separator " (inclusive). Between the markers of the message envelope is a decimal number with at least one digit. This digit then determines that *exactly* that number of 8-bit bytes are to be consumed as part of the token, without restriction. It is a lexical error for less than that number of bytes to be available.
4. Note that not all implementations of the ACC (see [FIPA00067]) will support the transparent transmission of 8-bit characters. It is the responsibility of the agent to ensure, by reference to internal API of the ACC, that a given channel is able to faithfully transmit the chosen message encoding.
5. Strings encoded in accordance with [ISO2022] may contain characters which are otherwise not permitted in the definition of `Word`. These characters are ESC (0x1B), SO (0x0E) and SI (0x0F). This is due to the complexity that would result from including the full [ISO2022] grammar in the above EBNF description. Hence, despite the basic description above, a word may contain any well-formed [ISO2022] encoded character, other (representations of) parentheses, spaces, or the # character. Strings must be enclosed between quote symbols. If the quote symbol itself needs to be part of the String, then it must be escaped by a \ character.
6. The format for time tokens is defined in section 3.10.
7. An agent is represented by its agent-identifier using the standard format from [FIPA00023].

### 687 3.10 Representation of Time

688 Time tokens are based on [ISO8601], with extension for relative time and millisecond durations. Time expressions may  
689 be absolute, or relative. Relative times are distinguished by the sign character + or - appearing as the first character in  
690 the token. If no type designator is given, the local time zone is then used. The type designator for UTC is the character  
691 z; UTC is preferred to prevent time zone ambiguities. Note that years must be encoded in four digits. As an example,  
692 8:30 am on 15th April, 1996 local time would be encoded as:

693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703

```
19960415T083000000
```

The same time in UTC would be:

```
19960415T083000000Z
```

while one hour, 15 minutes and 35 milliseconds from now would be:

```
+00000000T011500035
```



## 704 4 Reduced Expressivity Subsets of FIPA SL

705 The FIPA SL definition given above is a very expressive language, but for some agent communication tasks it is  
 706 unnecessarily powerful. This expressive power has an implementation cost to the agent and introduces problems of the  
 707 decidability of modal logic. To allow simpler agents, or agents performing simple tasks, to do so with minimal  
 708 computational burden, this section introduces semantic and syntactic subsets of the full FIPA SL content language for  
 709 use by the agent when it is appropriate or desirable to do so. These subsets are defined by the use of profiles, that is,  
 710 statements of restriction over the full expressive power of FIPA SL. These profiles are defined in increasing order of  
 711 expressivity as FIPA-SL0, FIPA-SL1 and FIPA-SL2.

712  
 713 Note that these subsets of FIPA SL, with additional ontological commitments (that is, the definition of domain predicates  
 714 and constants) are used in other FIPA specifications.  
 715

### 716 4.1 FIPA SL0: Minimal Subset

717 Profile 0 is denoted by the normative constant `fipa-sl0` in the `language` parameter of an ACL message. Profile 0 of  
 718 FIPA SL is the minimal subset of the FIPA SL content language. It allows the representation of actions, the  
 719 determination of the result a term representing a computation, the completion of an action and simple binary  
 720 propositions. The following defines the FIPA SL0 grammar:

```

721 Content           = "(" ContentExpression+ ")".
722
723 ContentExpression = ActionExpression
724                  | Proposition.
725
726 Proposition       = Wff.
727
728 Wff               = AtomicFormula
729                  | "(" ActionOp ActionExpression ")".
730
731 AtomicFormula     = PropositionSymbol
732                  | "(" "result"      Term Term ")"
733                  | "(" PredicateSymbol Term+ ")"
734                  | "true"
735                  | "false".
736
737 ActionOp          = "done".
738
739 Term              = Constant
740                  | Set
741                  | Sequence
742                  | FunctionalTerm
743                  | ActionExpression.
744
745 ActionExpression  = "(" "action" Agent Term ")".
746
747 FunctionalTerm    = "(" FunctionSymbol Term* ")"
748                  | "(" FunctionSymbol Parameter* ")".
749
750 Parameter         = ParameterName ParameterValue.
751
752 ParameterValue    = Term.
753
754 Agent             = Term.
755
756 FunctionSymbol     = String.
757
758 PropositionSymbol  = String.
759
760 PredicateSymbol   = String.
761
```

```

762
763 Constant          = NumericalConstant
764                   | String
765                   | DateTime.
766
767 Set                = "(" "set" Term* ")".
768
769 Sequence           = "(" "sequence" Term* ")".
770
771 NumericalConstant = Integer
772                   | Float.
773

```

774 The same lexical definitions described in Section 2.1 apply for FIPA SL0.  
775

## 776 4.2 FIPA SL1: Propositional Form

777 Profile 1 is denoted by the normative constant `fipa-sl1` in the `language` parameter of an ACL message. Profile 1 of  
778 FIPA SL extends the minimal representational form of FIPA SL0 by adding Boolean connectives to represent  
779 propositional expressions. The following defines the FIPA SL1 grammar:

```

780
781 Content            = "(" ContentExpression+ ")".
782
783 ContentExpression = ActionExpression
784                   | Proposition.
785
786 Proposition        = Wff.
787
788 Wff                = AtomicFormula
789                   | "(" UnaryLogicalOp Wff ")"
790                   | "(" BinaryLogicalOp Wff Wff ")"
791                   | "(" ActionOp ActionExpression ")".
792
793 UnaryLogicalOp     = "not".
794
795 BinaryLogicalOp    = "and"
796                   | "or".
797
798 AtomicFormula      = PropositionSymbol
799                   | "(" "result" Term Term ")"
800                   | "(" PredicateSymbol Term+ ")"
801                   | "true"
802                   | "false".
803
804 ActionOp           = "done".
805
806 Term               = Constant
807                   | Set
808                   | Sequence
809                   | FunctionalTerm
810                   | ActionExpression.
811
812 ActionExpression   = "(" "action" Agent Term ")".
813
814 FunctionalTerm     = "(" FunctionSymbol Term* ")"
815                   | "(" FunctionSymbol Parameter* ")".
816
817 Parameter          = ParameterName ParameterValue.
818
819 ParameterValue     = Term.
820
821 Agent              = Term.
822

```

```

823 FunctionSymbol      = String.
824
825 PropositionSymbol   = String.
826
827 PredicateSymbol     = String.
828
829 Constant            = NumericalConstant
830                    | String
831                    | DateTime.
832
833 Set                 = "(" "set" Term* ")".
834
835 Sequence            = "(" "sequence" Term* ")".
836
837 NumericalConstant   = Integer
838                    | Float.
839

```

840 The same lexical definitions described in Section 2.1 apply for FIPA SL1.

841

### 842 4.3 FIPA SL2: Decidability Restrictions

843 Profile 2 is denoted by the normative constant `fipa-sl2` in the `language` parameter of an ACL message. Profile 2 of  
844 FIPA SL allows first order predicate and modal logic, but is restricted to ensure that it must be decidable. Well-known  
845 effective algorithms exist that can derive whether or not an FIPA SL2 Wff is a logical consequence of a set of Wffs (for  
846 instance KSAT and Monadic). The following defines the FIPA SL2 grammar:

847

```

848 Content              = "(" ContentExpression+ ")".
849
850 ContentExpression    = IdentifyingExpression
851                    | ActionExpression
852                    | Proposition.
853
854 Proposition          = PrenexExpression.
855
856 Wff                  = AtomicFormula
857                    | "(" UnaryLogicalOp Wff ")"
858                    | "(" BinaryLogicalOp Wff Wff ")"
859                    | "(" ModalOp Agent PrenexExpression ")"
860                    | "(" ActionOp ActionExpression ")"
861                    | "(" ActionOp ActionExpression PrenexExpression ")".
862
863 UnaryLogicalOp       = "not".
864
865 BinaryLogicalOp      = "and"
866                    | "or"
867                    | "implies"
868                    | "equiv".
869
870 AtomicFormula        = PropositionSymbol
871                    | "(" "=" TermOrIE TermOrIE ")"
872                    | "(" "result" TermOrIE TermOrIE ")"
873                    | "(" PredicateSymbol TermOrIE+ ")"
874                    | "true"
875                    | "false".
876
877 PrenexExpression     = UnivQuantExpression
878                    | ExistQuantExpression
879                    | Wff.
880
881 UnivQuantExpression  = "(" "forall" Variable Wff ")"
882                    | "(" "forall" Variable UnivQuantExpression ")"
883                    | "(" "forall" Variable ExistQuantExpression ")".

```

```

884
885 ExistQuantExpression = "(" "exists" Variable Wff ")"
886                       | "(" "exists" Variable ExistQuantExpression ")".
887
888 TermOrIE              = Term
889                       | IdentifyingExpression.
890
891 Term                  = Variable
892                       | FunctionalTerm
893                       | ActionExpression
894                       | Constant
895                       | Sequence
896                       | Set.
897
898 IdentifyingExpression = "(" ReferentialOp TermOrIE Wff ")".
899
900 ReferentialOp        = "iota"
901                       | "any"
902                       | "all".
903
904 FunctionalTerm       = "(" FunctionSymbol TermOrIE* ")"
905                       | "(" FunctionSymbol Parameter* ")".
906
907 Parameter            = ParameterName ParameterValue.
908
909 ParameterValue       = TermOrIE.
910
911 ActionExpression     = "(" "action" Agent TermOrIE ")"
912                       | "(" "|" ActionExpression ActionExpression ")"
913                       | "(" ";" ActionExpression ActionExpression ")".
914
915 Variable             = VariableIdentifier.
916
917 Agent                = TermOrIE.
918
919 FunctionSymbol       = String.
920
921 Constant             = NumericalConstant
922                       | String
923                       | DateTime.
924
925 ModalOp              = "B"
926                       | "U"
927                       | "PG"
928                       | "I".
929
930 ActionOp             = "feasible"
931                       | "done".
932
933 PropositionSymbol    = String.
934
935 PredicateSymbol      = String.
936
937 Set                  = "(" "set" TermOrIE* ")".
938
939 Sequence             = "(" "sequence" TermOrIE* ")".
940
941 NumericalConstant    = Integer
942                       | Float.
943
944

```

The same lexical definitions described in Section 2.1 apply for FIPA SL2.

946

947 The `Wff` production of FIPA SL2 no longer directly contains the logical quantifiers, but these are treated separately to  
 948 ensure only prefixed quantified formulas, such as:

```
949
950 (forall ?x1
951   (forall ?x2
952     (exists ?y1
953       (exists ?y2
954         (Phi ?x1 ?x2 ?y1 ?y2))))))
```

955  
 956 Where `(Phi ?x1 ?x2 ?y1 ?y2)` does not contain any quantifier.

957  
 958 The grammar of FIPA SL2 still allows for quantifying-in inside modal operators. For example, the following formula is  
 959 still admissible under the grammar:

```
960
961 (forall ?x1
962   (or
963     (B i (p ?x1))
964     (B j (q ?x1))))
```

965  
 966 It is not clear that formulae of this kind are decidable. However, changing the grammar to express this context  
 967 sensitivity would make the EBNF form above essentially unreadable. Thus, the following additional mandatory  
 968 constraint is placed on well-formed content expressions using FIPA SL2: Within the scope of an `SLModalOperator`  
 969 only closed formulas are allowed, that is, formulas without free variables.

970

971

## 5 References

972

[FIPA00023] FIPA Agent Management Specification. Foundation for Intelligent Physical Agents, 2000.

973

<http://www.fipa.org/specs/fipa00023/>

974

[FIPA00037] FIPA Agent Communication Language Overview. Foundation for Intelligent Physical Agents, 2000.

975

<http://www.fipa.org/specs/fipa00037/>

976

[ISO8601] Date Elements and Interchange Formats, Information Interchange-Representation of Dates and Times.

977

International Standards Organisation, 1998.

978

<http://www.iso.ch/cate/d15903.html>

979

980 **6 Informative Annex A — Syntax and Lexical Notation**

981 The syntax is expressed in standard EBNF format. For completeness, the notation is given in *Table 2*.

982

Grammar rule component	Example
Terminal tokens are enclosed in double quotes	" ( "
Non terminals are written as capitalised identifiers	Expression
Square brackets denote an optional construct	[ ", " OptionalArg ]
Vertical bar denotes an alternative	Integer   Real
Asterisk denotes zero or more repetitions of the preceding expression	Digit *
Plus denotes one or more repetitions of the preceding expression	Alpha +
Parentheses are used to group expansions	( A   B ) *
Productions are written with the non-terminal name on the left-hand side, expansion on the right-hand side and terminated by a full stop	AnonTerminal = "an expansion".

983

**Table 2:** EBNF Rules

984

985

986 Some slightly different rules apply for the generation of lexical tokens. Lexical tokens use the same notation as above,  
 987 with the exceptions noted in *Table 3*.

988

Lexical rule component	Example
Square brackets enclose a character set	["a", "b", "c"]
Dash in a character set denotes a range	["a" - "z"]
Tilde denotes the complement of a character set if it is the first character	[~ "(, ")"]
Post-fix question-mark operator denotes that the preceding lexical expression is optional (may appear zero or one times)	["0" - "9"]? ["0" - "9"]

989

**Table 3:** Lexical Rules

990

991

## 992 7 Informative Annex B — ChangeLog

### 993 7.1 2002/11/01 - version H by TC X2S

994	Entire document:	Fixed bugs in the examples, by adding quotes and converting symbols into lower case
995	Entire document:	Added new non-terminal symbol <code>TermOrIE</code> and replaced all occurrences of <code>Term</code> with
996		<code>TermOrIE</code>
997	Page 2, line 72:	Added symbol identifying <code>fipa-sl</code> content language
998	<b>Page 2, lines 104-112:</b>	<b>Removed superfluous binary term operators</b>
999	<b>Page 3, lines 139-149:</b>	<b>Removed superfluous functional term operators</b>
1000	<b>Page 3, lines 180-184:</b>	<b>Removed superfluous arithmetic operators</b>
1001	<b>Page 4, line 224:</b>	<b>Added optional <code>sign</code> symbol to represent relative time</b>
1002	Pages 6, lines 342-373:	Removed description of superfluous equality operators
1003	Page 8, line 398:	Added note on interpretation of <code>iota</code> identifying expression
1004	Page 8, line 406:	Added note on interpretation of <code>iota</code> identifying expression
1005	Page 9, line 488 :	Added note on interpretation of <code>any</code> identifying expression
1006	Page 9, line 494:	Improved the definition of <code>any</code> identifying expression
1007	Page 9, line 497:	Improved the definition of <code>any</code> identifying expression
1008	Page 10, line 556:	Added note on interpretation of <code>all</code> identifying expression
1009	<b>Page 11, line 619:</b>	<b>Added requirement on encoding functional terms</b>
1010	Page 12, line 639:	Removed Table 1 on description of superfluous functional operators
1011	Page 12, lines 660-662:	Removed ambiguity in representing communicative acts in SL
1012	Page 12, line 664:	Added description of the actor of an ACL Message
1013	Page 13, lines 672-674:	Removed section on agent identifiers
1014	Page 13, lines 375-380:	Extended the section on Numerical Constants to incorporate more details on Grammar Rules
1015	Page 13, lines 682-692 :	Extended the section on Date and Time Constants to add a description of relative time
1016		